拦截器（Interceptor）和过滤器（Filter）的区别

Spring的Interceptor(拦截器)与Servlet的Filter有相似之处，比如二者都是AOP编程思想的体现，都能实现权限检查、日志记录等。不同的是：

Filter Interceptor Summary

Filter 接口定义在 javax.servlet 包中 接口 HandlerInterceptor 定义在org.springframework.web.servlet 包中

Filter 定义在 web.xml 中

Filter在只在 Servlet 前后起作用。Filters 通常将 请求和响应（request/response） 当做黑盒子，Filter 通常不考虑servlet 的实现。 拦截器能够深入到方法前后、异常抛出前后等，因此拦截器的使用具有更大的弹性。允许用户介入（hook into）请求的生命周期，在请求过程中获取信息，Interceptor 通常和请求更加耦合。 在Spring构架的程序中，要优先使用拦截器。几乎所有 Filter 能够做的事情， interceptor 都能够轻松的实现

Filter 是 Servlet 规范规定的。 而拦截器既可以用于Web程序，也可以用于Application、Swing程序中。 使用范围不同

Filter 是在 Servlet 规范中定义的，是 Servlet 容器支持的。 而拦截器是在 Spring容器内的，是Spring框架支持的。 规范不同

Filter 不能够使用 Spring 容器资源 拦截器是一个Spring的组件，归Spring管理，配置在Spring文件中，因此能使用Spring里的任何资源、对象，例如 Service对象、数据源、事务管理等，通过IoC注入到拦截器即可 Spring 中使用 interceptor 更容易

Filter 是被 Server(like Tomcat) 调用 Interceptor 是被 Spring 调用 因此 Filter 总是优先于 Interceptor 执行

拦截器（Interceptor）和过滤器（Filter）的执行顺序

过滤前-拦截前-Action处理-拦截后-过滤后

拦截器（Interceptor）使用

interceptor 的执行顺序大致为：

请求到达 DispatcherServlet

DispatcherServlet 发送至 Interceptor ，执行 preHandle

请求达到 Controller

请求结束后，postHandle 执行

Spring 中主要通过 HandlerInterceptor 接口来实现请求的拦截，实现 HandlerInterceptor 接口需要实现下面三个方法：

preHandle() – 在handler执行之前，返回 boolean 值，true 表示继续执行，false 为停止执行并返回。

postHandle() – 在handler执行之后, 可以在返回之前对返回的结果进行修改

afterCompletion() – 在请求完全结束后调用，可以用来统计请求耗时等等

Filter有如下几个用处。

在HttpServletRequest到达Servlet之前，拦截客户的HttpServletRequest。

根据需要检查HttpServletRequest，也可以修改HttpServletRequest头和数据。

在HttpServletResponse到达客户端之前，拦截HttpServletResponse。

根据需要检查HttpServletResponse，也可以修改HttpServletResponse头和数据。

Filter有如下几个种类。

用户授权的Filter：Filter负责检查用户请求，根据请求过滤用户非法请求。

日志Filter：详细记录某些特殊的用户请求。

负责解码的Filter:包括对非标准编码的请求解码。

能改变XML内容的XSLT Filter等。

Filter可以负责拦截多个请求或响应；一个请求或响应也可以被多个Filter拦截。

创建Filter必须实现javax.servlet.Filter接口，在该接口中定义了如下三个方法。

void init(FilterConfig config):用于完成Filter的初始化。

void destory():用于Filter销毁前，完成某些资源的回收。

void doFilter(ServletRequest request,ServletResponse response,FilterChain chain):实现过滤功能，该方法就是对每个请求及响应增加的额外处理。该方法可以实现对用户请求进行预处理(ServletRequest request)，也可实现对服务器响应进行后处理(ServletResponse response)—它们的分界线为是否调用了chain.doFilter(),执行该方法之前，即对用户请求进行预处理；执行该方法之后，即对服务器响应进行后处理。

过滤器拦截到响应url的请求后会先执行doFilter()方法中chain.doFilter()之前的代码，然后执行下一个过滤器或者servelt。紧接着执行chain.doFilter()之后的代码。

package com.rskd\_yswb.lib.filter;

import javax.servlet.\*;

import java.io.IOException;

public class HttpRequestAndResponseFilter implements Filter {

private FilterConfig filterConfig;

public void destroy() {

}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {

resp.setCharacterEncoding(this.filterConfig.getInitParameter("encoding"));

System.out.println("HttpRequestAndResponseFilterBefore");

chain.doFilter(req, resp);

System.out.println("HttpRequestAndResponseFilterAfter");

}

public void init(FilterConfig config) throws ServletException {

this.filterConfig = config;

}

}

package com.rskd\_yswb.lib.filter;

import javax.servlet.\*;

import java.io.IOException;

public class Test2Filter implements Filter {

public void destroy() {

}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {

System.out.println("Test2FilterBefore");

chain.doFilter(req, resp);

System.out.println("Test2FilterAfter");

}

public void init(FilterConfig config) throws ServletException {

}

}package com.rskd\_yswb.servelt;

import com.rskd\_yswb.javabean.db.ConsumerEntity;

import com.rskd\_yswb.lib.db.DBConnection;

import com.rskd\_yswb.lib.db.GenerateSqlStatement;

import com.rskd\_yswb.lib.freemarker.FreeMarker;

import freemarker.template.Template;

import freemarker.template.TemplateException;

import org.apache.commons.dbutils.QueryRunner;

import org.apache.commons.dbutils.handlers.BeanMapHandler;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import javax.servlet.annotation.WebServlet;

import java.io.IOException;

import java.io.Writer;

import java.sql.Connection;

import java.util.HashMap;

import java.util.Map;

public class IndexServlet extends javax.servlet.http.HttpServlet {

private static Logger logger = LogManager.getLogger(IndexServlet.class);

protected void doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) throws javax.servlet.ServletException, IOException {

doGet(request, response);

}

protected void doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) throws javax.servlet.ServletException, IOException {

String sql = new GenerateSqlStatement().select("\*").from("consumer").get();

try {

Map<String, ConsumerEntity> consumer = new QueryRunner().query(DBConnection.getConnection(), sql, new BeanMapHandler<String, ConsumerEntity>(ConsumerEntity.class, "username"));

} catch (Exception e) {

logger.debug("IndexServlet Error:" + e.getMessage());

}

Map<String, Object> root = new HashMap<String, Object>();

try {

FreeMarker.getFreemarker().getTemplate("index.ftl").process(root, response.getWriter());

System.out.println("请求处理中");

} catch (TemplateException e) {

logger.debug("IndexServlet Error:" + e.getMessage());

}

}

}

<web-app>

<filter>

<filter-name>HttpRequestAndResponseFilter</filter-name>

<filter-class>com.rskd\_yswb.lib.filter.HttpRequestAndResponseFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf8</param-value>

</init-param>

</filter>

<filter>

<filter-name>Test2Filter</filter-name>

<filter-class>com.rskd\_yswb.lib.filter.Test2Filter</filter-class>

</filter>

<filter-mapping>

<filter-name>HttpRequestAndResponseFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>Test2Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<servlet>

<servlet-name>IndexServlet</servlet-name>

<servlet-class>com.rskd\_yswb.servelt.IndexServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>IndexServlet</servlet-name>

<url-pattern>/index</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>IndexServlet</servlet-name>

<url-pattern></url-pattern>

</servlet-mapping>

</web-app>

filter的执行顺序如同web.xml中的配置的顺序，在浏览器输入http://localhost:8080/index 控制台将打印如下内容：

HttpRequestAndResponseFilterBefore

Test2FilterBefore

请求处理中

Test2FilterAfter

HttpRequestAndResponseFilterAfter

Spring中定义filter

@Component  
public class TimeFilter implements Filter {  
  
 private Logger logger = LoggerFactory.*getLogger*(this.getClass());  
  
 @Override  
 public void init(FilterConfig filterConfig) throws ServletException {  
 logger.info("my timeFilter init!");  
 }  
  
 @Override  
 public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws IOException, ServletException {  
 logger.info("my timeFilter start!");  
 long startTime = new Date().getTime();  
 //执行下一个过滤器  
 //在没有新的filter的情况下，doFilter（）返回当前的请求  
 filterChain.doFilter(request,response);  
 logger.info("耗时:" + (new Date().getTime() - startTime));  
 logger.info("my timeFilter end!");  
 }  
  
 @Override  
 public void destroy() {  
 logger.info("my timeFilter destroy!");  
 }  
}

继承filter，并加上@component注解即可

*如何将第三方中没用component注解的filter加入到springboot的项目中的过滤器链上*

使用配置类

@Configuration

package com.cxy.security.cxysecuritydemo.config;  
  
import com.cxy.security.cxysecuritydemo.filter.TimeFilter;  
import org.springframework.boot.web.servlet.FilterRegistrationBean;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*\*  
 \* Created by Caoxingyun on 2019/03/15  
 \* 何将第三方中没用component注解的filter加入到springboot的项目中的过滤器链上  
 \* 配置类  
 \*/*@Configuration  
public class WebConfig {  
  
 // bean配置类似于在web.xml中手动配置filter  
//  
// <filter-mapping>  
// <filter-name>Test2Filter</filter-name>  
// <url-pattern>/\*</url-pattern>  
// </filter-mapping>  
 @Bean  
 public FilterRegistrationBean timeFilter(){  
 FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean();  
 TimeFilter timeFilter = new TimeFilter();  
 filterRegistrationBean.setFilter(timeFilter);  
 //配置过滤器的拦截路径  
 List<String> urls = new ArrayList<>();  
 urls.add("\*");  
 filterRegistrationBean.setUrlPatterns(urls);  
 return FilterRegistrationBean  
 }  
}